In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
df = pd.read_csv('data/faults.csv')
df.head(20)
```

Out[ ]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter	Sum_of_Luminosity	Minimum_of
0	42	50.0	270900	270944	267	17	44	24220	
1	645	651.0	2538079	2538108	108	10	30	11397	
2	829	835.0	1553913	1553931	71	8	19	7972	
3	853	860.0	369370	369415	176	13	45	18996	
4	1289	1306.0	498078	498335	2409	60	260	246930	
5	430	441.0	100250	100337	630	20	87	62357	
6	413	446.0	138468	138883	9052	230	432	1481991	
7	190	200.0	210936	210956	132	11	20	20007	
8	330	343.0	429227	429253	264	15	26	29748	
9	74	90.0	779144	779308	1506	46	167	180215	
10	106	118.0	813452	813500	442	13	48	50393	
11	505	515.0	106604	106668	284	42	69	31062	
12	46	58.0	179258	179312	480	15	54	61966	
13	581	590.0	230644	230704	433	22	60	38917	
14	451	466.0	368143	368208	728	30	68	69258	
15	669	684.0	491552	491684	1097	59	133	119540	
16	156	192.0	713788	714056	5044	167	282	570911	
17	90	104.0	751059	751132	552	38	76	59750	
18	82	89.0	844704	844729	137	8	25	14907	
19	1601	1613.0	21349	21376	209	15	27	24807	

20 rows × 28 columns

Replacing zeros with null

In [ ]:
```
df.duplicated().any()
```

Out[ ]:
```
False
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
X_Minimum                0
X_Maximum               24
Y_Minimum                0
Y_Maximum                0
Pixels_Areas             0
X_Perimeter              0
Y_Perimeter              0
Sum_of_Luminosity        0
Minimum_of_Luminosity    0
Maximum_of_Luminosity    0
Length_of_Conveyer       0
TypeOfSteel_A300         0
TypeOfSteel_A400         0
Steel_Plate_Thickness   30
Edges_Index              0
Empty_Index             26
Square_Index             0
Outside_X_Index          0
Edges_X_Index            0
Edges_Y_Index            0
Outside_Global_Index     0
LogOfAreas               0
Log_X_Index              0
Log_Y_Index              0
Orientation_Index        0
Luminosity_Index         0
SigmoidOfAreas           0
target                   0
dtype: int64
```

In [ ]:
```
df.describe()
```

Out[ ]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter	Sum_of_Luminosity	M
count	1941.000000	1917.000000	1.941000e+03	1.941000e+03	1941.000000	1941.000000	1941.000000	1.941000e+03	
mean	571.136012	613.564945	1.650685e+06	1.650739e+06	1893.878413	111.855229	82.965997	2.063121e+05	
std	520.690671	496.108846	1.774578e+06	1.774590e+06	5168.459560	301.209187	426.482879	5.122936e+05	
min	0.000000	4.000000	6.712000e+03	6.724000e+03	2.000000	2.000000	1.000000	2.500000e+02	
25%	51.000000	192.000000	4.712530e+05	4.712810e+05	84.000000	15.000000	13.000000	9.522000e+03	
50%	435.000000	458.000000	1.204128e+06	1.204136e+06	174.000000	26.000000	25.000000	1.920200e+04	
75%	1053.000000	1066.000000	2.183073e+06	2.183084e+06	822.000000	84.000000	83.000000	8.301100e+04	
max	1705.000000	1713.000000	1.298766e+07	1.298769e+07	152655.000000	10449.000000	18152.000000	1.159141e+07	

8 rows × 27 columns

In [ ]:
```
data = df.values
X = data[:, :-1]
y = data[:, -1]
pd.DataFrame(X)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	17	18	19	20	21	22	23
0	42	50.0	270900	270944	267	17	44	24220	76	108	...	0.0047	0.4706	1.0	1.0	2.4265	0.9031	1.6435
1	645	651.0	2538079	2538108	108	10	30	11397	84	123	...	0.0036	0.6	0.9667	1.0	2.0334	0.7782	1.4624
2	829	835.0	1553913	1553931	71	8	19	7972	99	125	...	0.0037	0.75	0.9474	1.0	1.8513	0.7782	1.2553
3	853	860.0	369370	369415	176	13	45	18996	99	126	...	0.0052	0.5385	1.0	1.0	2.2455	0.8451	1.6532
4	1289	1306.0	498078	498335	2409	60	260	246930	37	126	...	0.0126	0.2833	0.9885	1.0	3.3818	1.2305	2.4099
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1936	249	277.0	325780	325796	273	54	22	35033	119	141	...	0.0206	0.5185	0.7273	0.0	2.4362	1.4472	1.2041
1937	144	175.0	340581	340598	287	44	24	34599	112	133	...	0.0228	0.7046	0.7083	0.0	2.4579	1.4914	1.2305
1938	145	174.0	386779	386794	292	40	22	37572	120	140	...	0.0213	0.725	0.6818	0.0	2.4654	1.4624	1.1761
1939	137	170.0	422497	422528	419	97	47	52715	117	140	...	0.0243	0.3402	0.6596	0.0	2.6222	1.5185	1.4914
1940	1261	1281.0	87951	87967	103	26	22	11682	101	133	...	0.0147	0.7692	0.7273	0.0	2.0128	1.301	1.2041

1941 rows × 27 columns

In [ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=1)
```

In [ ]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(1552, 27)
(389, 27)
```

Mean - Empty_Index, (column:15)

Mode - Steel_Plate_Thickness, (column:13)

Median - X_Maximum (column:1)

In [ ]:
```
mean_imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
median_imputer = SimpleImputer(missing_values=np.nan, strategy='median')
mode_imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
min_max_scaler = MinMaxScaler(feature_range=(0,1))
```

In [ ]:
```
X_train[:, (15,)] = mean_imputer.fit_transform(X_train[:, (15,)])
X_train[:, (1,)] = median_imputer.fit_transform(X_train[:, (1,)])
X_train[:, (13,)] = mode_imputer.fit_transform(X_train[:, (13,)])

X_test[:, (15,)] = mean_imputer.transform(X_test[:, (15,)])
X_test[:, (1,)] = median_imputer.transform(X_test[:, (1,)])
X_test[:, (13,)] = mode_imputer.transform(X_test[:, (13,)])
```

In [ ]:
```
pd.DataFrame(X_train).isna().any()
```

Out[ ]:
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
dtype: bool
```

In [ ]:
```
pd.DataFrame(X_train)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	17	18	19	20	21	22	23
0	1192	1205.0	521912	521925	101	22	13	11601	108	125	...	0.0096	0.5909	1.0	0.5	2.0043	1.1139	1.1139
1	129	157.0	86408	86427	276	39	26	33858	115	135	...	0.0206	0.7179	0.7308	0.0	2.4409	1.4472	1.2787
2	228	253.0	3458746	3458793	561	57	50	60112	61	124	...	0.0184	0.4386	0.94	1.0	2.749	1.3979	1.6721
3	981	992.0	467888	467902	102	16	14	12417	115	132	...	0.0081	0.6875	1.0	1.0	2.0086	1.0414	1.1461
4	371	385.0	2794886	2794911	172	24	25	19549	105	125	...	0.0103	0.5833	1.0	1.0	2.2355	1.1461	1.3979
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1547	209	259.0	9649727	9649771	1182	87	71	130201	96	127	...	0.0368	0.5747	0.6197	0.0	3.0726	1.699	1.6435
1548	362	388.0	2839568	2839625	1015	49	59	105560	76	132	...	0.0191	0.5306	0.9661	1.0	3.0065	1.415	1.7559
1549	239	269.0	276029	276047	299	51	22	37820	116	140	...	0.0221	0.5882	0.8182	0.0	2.4757	1.4771	1.2553
1550	119	127.0	1172603	1172632	128	22	30	15896	111	140	...	0.0059	0.3636	0.9667	1.0	2.1072	0.9031	1.4624
1551	586	616.0	1862942	1862979	767	42	37	55228	12	134	...	0.0185	0.7143	1.0	1.0	2.8848	1.4771	1.5682

1552 rows × 27 columns

Applying Label encoding on target column

In [ ]:
```
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

Applying min-max scaling on all independant variables

In [ ]:
```
X_train = min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)
```

In [ ]:
```
pd.DataFrame(X_train)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	17	18
0	0.699120	0.702750	0.039689	0.039689	0.000649	0.001914	0.000661	0.000979	0.532020	0.407407	...	0.013041	0.584923
1	0.075660	0.089526	0.006139	0.006140	0.001795	0.003542	0.001377	0.002899	0.566502	0.453704	...	0.030752	0.713778
2	0.133724	0.145699	0.265931	0.265933	0.003662	0.005265	0.002700	0.005164	0.300493	0.402778	...	0.027210	0.430398
3	0.575367	0.578116	0.035527	0.035527	0.000655	0.001340	0.000716	0.001049	0.566502	0.439815	...	0.010626	0.682934
4	0.217595	0.222937	0.214790	0.214790	0.001114	0.002106	0.001322	0.001665	0.517241	0.407407	...	0.014168	0.577212
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1547	0.122581	0.149210	0.742859	0.742860	0.007730	0.008136	0.003857	0.011211	0.472906	0.416667	...	0.056835	0.568486
1548	0.212317	0.224693	0.218232	0.218235	0.006636	0.004499	0.003195	0.009085	0.374384	0.439815	...	0.028337	0.523742
1549	0.140176	0.155061	0.020747	0.020748	0.001946	0.004690	0.001157	0.003241	0.571429	0.476852	...	0.033167	0.582183
1550	0.069795	0.071972	0.089816	0.089817	0.000825	0.001914	0.001598	0.001349	0.546798	0.476852	...	0.007084	0.354302
1551	0.343695	0.358104	0.142996	0.142998	0.005011	0.003829	0.001983	0.004743	0.059113	0.449074	...	0.027371	0.710126

1552 rows × 27 columns

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, f1_score
```

Naive Bayes Classification

In [ ]:
```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

Out[ ]:
```
GaussianNB()
```

In [ ]:
```
y_pred_gnb = gnb.predict(X_test)
```

In [ ]:
```
accuracy_gnb = accuracy_score(y_test, y_pred_gnb)
precision_gnb = precision_score(y_test, y_pred_gnb, average='weighted')
recall_gnb = recall_score(y_test, y_pred_gnb, average='weighted')
f1_gnb = f1_score(y_test, y_pred_gnb, average='weighted')
```

In [ ]:
```
cr_gnb = classification_report(y_test, y_pred_gnb, output_dict=True, zero_division='warn')
pd.DataFrame(cr_gnb)
```

Out[ ]:

	0	1	2	3	4	5	6	accuracy	macro avg	weighted avg
precision	0.475806	0.333333	0.942029	0.672727	0.295082	0.764706	0.871795	0.601542	0.622211	0.674784
recall	0.746835	0.800000	0.812500	0.268116	0.620690	1.000000	0.850000	0.601542	0.728306	0.601542
f1-score	0.581281	0.470588	0.872483	0.383420	0.400000	0.866667	0.860759	0.601542	0.633600	0.592891
support	79.000000	10.000000	80.000000	138.000000	29.000000	13.000000	40.000000	0.601542	389.000000	389.000000

KNN Classification

Selecting best value for n_neighbors using GridsearchCV

In [ ]:
```
knn_test = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(1, 40)}
#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn_test, param_grid, cv=5)
knn_gscv.fit(X_train, y_train)
```

Out[ ]:

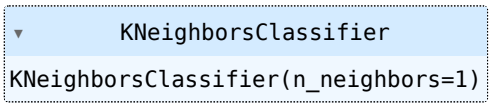
In [ ]:
```
knn_gscv.best_params_
```

Out[ ]:
```
{'n_neighbors': 1}
```

Here, 1 gives best result for n_neighbors

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
y_pred_knn = knn.predict(X_test)
```

In [ ]:
```
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn, average='weighted')
recall_knn = recall_score(y_test, y_pred_knn, average='weighted')
f1_knn = f1_score(y_test, y_pred_knn, average='weighted')
```

In [ ]:
```
cr_knn = classification_report(y_test, y_pred_knn, output_dict=True, zero_division='warn')
pd.DataFrame(cr_knn)
```

Out[ ]:

	0	1	2	3	4	5	6	accuracy	macro avg	weighted avg
precision	0.575000	0.857143	0.935897	0.636364	0.571429	0.923077	0.686275	0.699229	0.740741	0.701051
recall	0.582278	0.600000	0.912500	0.608696	0.551724	0.923077	0.875000	0.699229	0.721896	0.699229
f1-score	0.578616	0.705882	0.924051	0.622222	0.561404	0.923077	0.769231	0.699229	0.726355	0.698227
support	79.000000	10.000000	80.000000	138.000000	29.000000	13.000000	40.000000	0.699229	389.000000	389.000000

SVM Classification

In [ ]:
```
svm = SVC()
svm.fit(X_train, y_train)
```

Out[ ]:
```
SVC()
```

In [ ]:
```
y_pred_svm = svm.predict(X_test)
```

In [ ]:
```
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')
```

In [ ]:
```
cr_svm = classification_report(y_test, y_pred_svm, output_dict=True, zero_division='warn')
pd.DataFrame(cr_svm)
```

Out[ ]:

	0	1	2	3	4	5	6	accuracy	macro avg	weighted avg
precision	0.641026	1.000000	0.986111	0.639752	0.733333	0.764706	0.795455	0.732648	0.794340	0.747665
recall	0.632911	0.200000	0.887500	0.746377	0.379310	1.000000	0.875000	0.732648	0.674443	0.732648
f1-score	0.636943	0.333333	0.934211	0.688963	0.500000	0.866667	0.833333	0.732648	0.684779	0.726390
support	79.000000	10.000000	80.000000	138.000000	29.000000	13.000000	40.000000	0.732648	389.000000	389.000000

Comparing Classification Scores

In [ ]:
```
comparison_data = (('Naive Bayes', accuracy_gnb, precision_gnb, recall_gnb, f1_gnb),
                   ('K Nearest Neighbor', accuracy_knn, precision_knn, recall_knn, f1_knn),
                   ('Support Vector', accuracy_svm, precision_svm, recall_svm, f1_svm))
```

In [ ]:
```
pd.DataFrame(comparison_data, columns = ('Comparison algorithm','Accuracy','Precision','Recall','F-measure'))
```

Out[ ]:

	Comparison algorithm	Accuracy	Precision	Recall	F-measure
0	Naive Bayes	0.601542	0.674784	0.601542	0.592891
1	K Nearest Neighbor	0.699229	0.701051	0.699229	0.698227
2	Support Vector	0.732648	0.747665	0.732648	0.726390